COMP2017 9017 Assignment 1

Release: 18:00 Friday 3 March 2022 Sydney local time

Due: 23:59 Tuesday 14 March 2022 Sydney local time

Duration: 11 days

This assignment is worth 10% of your final assessment

Task description

In this assignment, we will develop a game controller that accommodates a two-player board game called *Gomoku*[1] *in the* ***Mist***. The controller should be programmed in the C programming language **without** using dynamic memory. And always ensure that no memory errors occur.

You are encouraged to ask questions on Ed[2]. Make sure your question post is of "**Question**" post type and is under "**Assignment**" category → "**A1**" subcategory → "**Debugging**"/"**Spec**"/"**General**" sub-subcategory. As with any assignment, make sure that your work is your own[3], and that you do not share your code or solutions with other students.

Before attempting this assignment it would be a good idea to familiarise yourself with arrays, standard I/O and C strings. In particular, you may find the following functions helpful:

```
fscanf()
fgets()
```

Rules of the Game

Gomoku or Five in a Row is played on a 19×19 Go board (See Figure 1). The game starts with an empty board, and players **Black** and **White** alternate to place a stone of their colour on a grid point. **Black** goes first. The goal of the game is to create a line of five or more stones of your own colour, either horizontally, vertically or diagonally. The first player to achieve that goal wins (See Figure 2. If neither player creates five in a row when the board is full, then the game is a draw. The game finishes when a player wins, a player resigns or they draw.

[1] You may explore this Wikipedia page.
[2] https://edstem.org/au/courses/10466/discussion/
[3] Not GPT-3/4's, ChatGPT's or copilot's, etc.

Gomoku** in the **Mist is a new game based on **Gomoku**. It inherits all the rules from **Gomoku** and it introduces a game component called **Mist**. **Mist** blocks the visibility of both players and it covers the whole board **except** a 7×7 square **hole** (See Figure 3). Players can only see through the **hole** that is not covered by the **Mist**, however, the placement of stone is not blocked by the **Mist** at all. If the centre of the **hole** is so close to the border of the board that the **hole** cannot fit into the board, simply ignore the part that cannot fit.

Initially, the **hole** of the **Mist** locates at the centre of the board before the first move. The **Mist** (**hole**) changes after every placement of a stone. The position of **hole** depends on the previous placement of a stone. If a numerical coordinate system is used, the following rules apply:

$$x_{\text{mist}} = 1 + (5x_{\text{stone}}^2 + 3x_{\text{stone}} + 4) \,\%\, 19,$$
$$y_{\text{mist}} = 1 + (4y_{\text{stone}}^2 + 2y_{\text{stone}} - 4) \,\%\, 19.$$

$\%$ is the modulo operator.

$(x_{\text{mist}}, y_{\text{mist}})$ denotes the position of the **hole**'s centre, and $(x_{\text{stone}}, y_{\text{stone}})$ denotes the position of the previous placed stone.

To denote position on the board, a coordinate `<C><R>` can be specified. While it is natural to utilise a 19×19 purely numerical coordinate system, the Go style system is preferred. In particular, for a Go coordinate `<C><R>`, `<C>` is a column character ranging from `A` (the first English upper letter) to `S` (the nineteenth English upper letter); `<R>` is a row number ranging from `1` to `19`.

In a numerical coordinate system, columns are denoted by numbers instead of characters. In particular, the n-th English upper letter is equivalent to a number n. E.g. Column character `A` is equivalent to column number 1. See the `numerical` numbering in Figure 1.

Functionalities of the Game Controller

The Game Controller is a command prompt that supports the following commands:

```
who
term
resign
view
place <C><R>
history
```

Note that a new game should be automatically started when the game controller is launched, and the game controller should terminate when a game is finished or the controller is forced to terminate.

Commands are case-sensitive and whitespace-sensitive. The game controller will reject unspecified commands or known commands with unspecified trailing whitespaces or parameters. The only command that accepts a parameter is `place` and it can only be supplied with one parameter. The game controller should output `Invalid!` if it rejects the user input.

```
               19 . . . . . . . . . . . . . . . . . . .
               18 . . . . . . . . . . . . . . . . . . .
               17 . . . . . . . . . . . . . . . . . . .
               16 . . . . . . . . . . . . . . . . . . .
               15 . . . . . . . . . . . . . . . . . . .
               14 . . . . . . . . . . . . . . . . . . .
               13 . . . . . . . . . . . . . . . . . . .
               12 . . . . . . . . . . . . . . . . . . .
               11 . . . . . . . . . . . . . . . . . . .
               10 . . . . . . . . . . . . . . . . . . .
               9  . . . . . . . . . . . . . . . . . . .
               8  . . . . . . . . . . . . . . . . . . .
               7  . . . . . . . . . . . . . . . . . . .
               6  . . . . . . . . . . . . . . . . . . .
               5  . . . . . . . . . . . . . . . . . . .
               4  . . . . . . . . . . . . . . . . . . .
               3  . . . . . . . . . . . . . . . . . . .
               2  . . . . . . . . . . . . . . . . . . .
               1  . . . . . . . . . . . . . . . . . . .
                  A B C D E F G H I J K L M N O P Q R S
numerical:        1 2 3 4 5 6 7 8 9 ..................
```

Figure 1: An ASCII art of the Go board grid points with axes numbering

```
               19 # # # # # . . . . . . . . . . . . . .
               18 o o o o . . . . . . . . . . . . . . .
               17 . . . . . . . . . . . . . . . . . . .
               16 . . . . . . . . . . . . . . . . . . .
               15 . . . . . . . . . . . . . . . . . . .
               14 . . . . . . . . . . . . . . . . . . .
               13 . . . . . . . . . . . . . . . . . . .
               12 . . . . . . . . . . . . . . . . . . .
               11 . . . . . . . . . . . . . . . . . . .
               10 . . . . . . . . . . . . . . . . . . .
               9  . . . . . . . . . . . . . . . . . . .
               8  . . . . . . . . . . . . . . . . . . .
               7  . . . . . . . . . . . . . . . . . . .
               6  . . . . . . . . . . . . . . . . . . .
               5  . . . . . . . . . . . . . . . . . . .
               4  . . . . . . . . . . . . . . . . . . .
               3  . . . . . . . . . . . . . . . . . . .
               2  . . . . . . . . . . . . . . . . . . .
               1  . . . . . . . . . . . . . . . . . . .
                  A B C D E F G H I J K L M N O P Q R S
```

Figure 2: An ASCII art of the Go board when Black stone wins, where `#`, `o`, `.` represents Black stone, White stone and no stone (empty).

```
19 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
18 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
17 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
16 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
15 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
14 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
13 @ @ @ @ @ @ . . . . . . . @ @ @ @ @ @
12 @ @ @ @ @ @ . . . . . . . @ @ @ @ @ @
11 @ @ @ @ @ @ . . . . . . . @ @ @ @ @ @
10 @ @ @ @ @ @ . . . A . . . @ @ @ @ @ @
9  @ @ @ @ @ @ . . . . . . . @ @ @ @ @ @
8  @ @ @ @ @ @ . . . . . . . @ @ @ @ @ @
7  @ @ @ @ @ @ . . . . . . . @ @ @ @ @ @
6  @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
5  @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
4  @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
3  @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
2  @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
1  @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
   A B C D E F G H I J K L M N O P Q R S
```

Figure 3: An ASCII art of the Go board covered by the **Mist** except the 7×7 **hole**, where `@`, `.`, `A` represents the **Mist**, the **hole** and the centre of the **hole**.

`who`

`who` command can be executed to show who is the current player. One character will be displayed, either `B` for Black stone or `W` for White stone.

This command can be executed at any time.

`term`

`term` command forces the controller to terminate with exit code 1. This command can be executed at any time. This command will suppress history moves messages and the exit message.

`resign`

`resign` command can be executed by a player to resign from the game. This command will cause the current player to end the game and lose. The other player will win the game. This command can be executed at any time.

```
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49
```

Figure 4: An ASCII art of the 7×7 **hole** in Figure 3, where the numbers represent the output ordering of the **hole** in `view`.

`view`

`view` command can be executed by a player to view the current board. The controller will respond with a one-line output, consisting of two strings separated by a comma.

The first string denotes the centre coordinate of the 7×7 **hole** that is not covered by the **Mist**.

The second string denotes the status of grid points in the **hole**. That is, whether each point is occupied by a black stone, a white stone, not occupied or off-board. This string is the concatenation of rows' status (from the greatest row number 19 to the least row number 1), and each row starts with the lowest column (from `A` to `S`). See Figure 4.

`#`, `o`, `.`, `x` denotes black, white, empty and off-board.

If the **hole** is partially off-board, then the grid points that are off-board should be represented by `x`. See Example 4.

This command can be executed at any time.

`place <C><R>`

`place` command can be executed by a player to place a stone on the Go board according to the Go coordinate `<C><R>`. If the placement is successful, the controller will not respond with any output and silently alternate the player. However, if the given placement is invalid, then it will respond with a one-line error message without alternating the player. If the placement (e.g. `Z100`) cannot be recognised as a valid board coordinate, then it outputs `Invalid coordinate`. If the placement can be recognised as a valid board coordinate however the coordinate is occupied, then it outputs `Occupied coordinate`. This command can be executed at any time.

Placements in the **Mist** are allowed.

`history`

`history` command can be executed to obtain all history of moves in Go coordinates `<C><R>`. In particular, a one-line output formed by a sequence of the Go coordinates. For example, if Black first places a stone at `A1`, then White at `B2`, followed by Black at `C3` and finally White at `D4`, the history requested at this point should read `A1B2C3D4`. This command can be executed at any time.

End game conditions

A winner is decided when that player achieves a line of five stones with the same colour, either horizontally, vertically or diagonally, or the opposing player has `resign`ed the game. A message will be displayed

```
[Black|White] wins!
```

The game will also end when no more stones can be placed A message will be displayed

```
Wow, a tie!
```

At the end of the game, the `history` command should be called silently to display all the moves.

Finally, an exit message will be provided

```
Thank you for playing!
```

The controller will terminate with exit code 0 if a player wins or there is a tie.

`term` command will not lead to the end game.

Examples

User inputs are highlighted in green. Program outputs are not highlighted.

Example 1

```
who
B
term
```

Example 2

```
who
B
place A1
who
W
resign
Black wins!
A1
Thank you for playing!
```

Example 3

```
place A1
place B2
place C3
place D4
history
A1B2C3D4
term
```

Example 4

```
view
J10,.................................................
place A1
view
M3,..........................................xxxxxxx
place B2
view
L17,xxxxxxx..........................................
place C3
view
B1,xx.....xx..#..xx.o...xx#....xxxxxxxxxxxxxxxxxxxxx
place D4
view
B12,xx.....xx.....xx.....xx.....xx.....xx.....xx.....
history
A1B2C3D4
term
```

Example 5

```
place A1
place B1
place A2
place B2
place A3
place B3
place A4
place B4
place A5
Black wins!
A1B1A2B2A3B3A4B4A5
Thank you for playing!
```

Example 6

```
who
B
place Z1
Invalid coordinate
who
B
place A1
who
W
place A1
Occupied coordinate
who
W
place B122
Invalid coordinate
who
W
place B1
who
B
place A2
place B2
place A3
place B3
place A4
place B4
place A5
Black wins!
A1B1A2B2A3B3A4B4A5
Thank you for playing!
```

Submission details

You are encouraged to submit multiple times, but only your last submission before the deadline will be marked.

Writing your own test cases

We have provided you with some test cases but these do not test all the functionality described in the assignment. It is important that you thoroughly test your code by writing your own test cases.

You should place all of your test cases in the `tests/` directory. Ensure that each test case has the `.in` input file along with a corresponding `.out` output file. We recommend that the names of your test cases are descriptive so that you know what each is testing, e.g. `basic-game.in` and `tie-condition.in`. Please note that test case files must be located in `tests/` directory and follow the above naming convention. Otherwise, they will **NOT** be marked.

Restrictions

If your program does not adhere to these restrictions, your submission will receive **ZERO** for this assignment.

- No additional source code files of the game controller than what the scaffold provides
- No Variable Length Arrays (VLAs)
- No dynamic memory (calls to `*alloc*` functions)

Hint: Use compiler flags to eliminate the existence of VLAs. You may avoid including `stdlib.h` header file to avoid calling `*alloc*` functions.

Marking

A grade for this assignment is made where there is a submission that compiles.

8 **marks** are assigned based on automatic tests for the *correctness* of your program.
This component will also use hidden test cases to cover the assignment description.

2 **marks** are based on your submission of test cases, which considers coverage of input scenarios.

Additionally marks will be deducted if:

- excessive memory is used, e.g preallocation or over allocation of what is required.
- uses unjustifiable magic numbers.
- uses files, or mapped memory.

Warning: Any attempts to deceive the automatic marking will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment description, or your code is unnecessarily or deliberately obfuscated.

Working on your assignment

You can work on this assignment on your own computer or the lab machines. It is important that you continually back up your assignment files onto your own machine, external drives, and in the cloud. You are responsible for your assignment files to remain private to you, and not accessible by others.

You are encouraged to submit your assignment on Ed while you are in the process of completing it. By submitting you will obtain some feedback of your progress on the sample test cases provided.

If you have any questions about any C functions, then refer to the corresponding `man` pages. You can ask questions about this assignment on Ed. As with any assignment, make sure that your work is your own, and that you do not share your code or solutions with other students.

Where to start

Begin with simple entries and their operations.

- confirm you can process the line of input and recognise commands, integers (of multiple digits)
- make the basic Gomoku
- avoid implementing the **Mist** to begin with
- implement a single move to confirm the history is correct, then attempt with multiple
- check the simplest possible win condition as a starting point. e.g. five stones placed horizontally, vertically and diagonally. These trivial cases will check if you have arranged your 2D grid correctly.

Next, approach the addition of **Mist** and confirm your test data and operations of input meet your expectation.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

Changes

Any changes made to this document will be updated here.

- March 4th: Fix incorrect figures & typos and add clarifications. All major changes are highlighted in yellow.